Trace Theory and Systolic Computations

Martin Rem

Computer Science Department
California Institute of Technology

5239:TR:87

TRACE THEORY AND SYSTOLIC COMPUTATIONS

Martin Rem
Dept. of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, Netherlands

0. Introduction

We discuss a class of concurrent computations, or special-purpose computing engines, that may be characterized by

(i) they consist of regular arrangements of simple cells;
(ii) the arrangement consumes streams of input values and produces streams of output values;
(iii) the cells communicate with a fixed number of neighbor cells only;
(iv) the communication behaviors of the cells are independent of the values communicated.

Such arrangements are often referred to as systolic arrays [5]. Our computations, however, have a few other characteristics that are usually not found among systolic arrays:

(v) synchronization of cells is by message passing only;
(vi) each output value is produced as soon as all input values on which it depends have been consumed.

The formalism we use to discuss these computations is trace theory [4], [7], [8]. Section 1 is an introduction to trace theory, in which only those subjects are covered that are needed to understand the subsequent sections. Section 2, called Data Independence, addresses the question what it means that communication behaviors are independent of the values communicated. To express the simplicity of (the communication behaviors of) the cells we define in Section 3 the concept of conservative processes. The results of Sections 2 and 3 are assembled into a number of theorems that are used in Sections 4, 5, and 6. Each of these remaining sections discusses an illustrative example of a systolic computation: polynomial multiplication, cyclic encoding, and palindrome recognition.

1. Processes

This section is a trace-theoretic introduction to processes. A process is an abstraction of a mechanism, capturing the ways in which the mechanism can interact with its environment. A process is characterized by the set of events it can be involved in and by the possible orders in which these events can occur. Events are represented by *symbols*. Sets of

symbols are called *alphabets* and finite-length sequences of symbols are called *traces*. The set of all traces with symbols from alphabet A is denoted by A^*.

A *process* is a pair $\langle A, X \rangle$, where A is an alphabet and X is a non-empty prefix-closed set of traces with symbols from A:

$$X \subseteq A^*$$
$$X \neq \phi$$
$$X = \mathit{pref}(X)$$

where $\mathit{pref}(X)$ denotes set X extended with all the prefixes of traces in X:

$$\mathit{pref}(X) = \{t \in A^* \mid (\exists u : u \in A^* : tu \in X)\}$$

For process T we let $\mathbf{a}T$ denote its alphabet and $\mathbf{t}T$ its set of traces: $T = \langle \mathbf{a}T,\ \mathbf{t}T \rangle$.

An example of a process is

$$\langle \{a, b\},\ \{\varepsilon, a, ab, aba, abab, \ldots\} \rangle$$

(ε denotes the empty trace.) We call this process $SEM_1(a, b)$. Its trace set consists of all finite alternations of a and b that do not start with b:

$$SEM_1(a, b) = \langle \{a, b\}, \mathit{pref}(\{ab\}^*) \rangle$$

where, for X a set of traces, X^* denotes the set of all finite concatenations of traces in X.

The central operators of trace theory are projection and weaving. They are the formal counterparts of abstraction and composition respectively. The *projection* of trace t on alphabet A, denoted by $t^{\frown}A$, is obtained by removing from t all symbols that are not in A. We may write $t^{\frown}a$ for $t^{\frown}\{a\}$. We extend the definition of projection from traces to processes as follows:

$$T^{\frown}A = \langle \mathbf{a}T \cap A, \{t \mid (\exists u : u \in \mathbf{t}T : u^{\frown}A = t)\} \rangle$$

For example,

$$SEM_1(a, b)^{\frown}a = \langle \{a\}, \{a\}^* \rangle$$

The *weave* of processes T and U, denoted by $T\ \mathbf{w}\ U$, is defined by

$$T\ \mathbf{w}\ U = \langle \mathbf{a}T \cup \mathbf{a}U,\ \{t \in (\mathbf{a}T \cup \mathbf{a}U)^* \mid t^{\frown}\mathbf{a}T \in \mathbf{t}T \wedge t^{\frown}\mathbf{a}U \in \mathbf{t}U\} \rangle$$

For example,

$$SEM_1(a, b)\ \mathbf{w}\ SEM_1(b, a) = \langle \{a, b\}, \{\varepsilon\} \rangle$$

and

$$SEM_1(a,b) \mathbin{\mathbf{w}} SEM_1(a,c) = \langle \{a,b,c\}, pref(\{abc, acb\}^*) \rangle$$

Let $t \in \mathbf{t}T$. The *successor set* of t in T, denoted by $S(t,T)$, is defined by

$$S(t,T) = \{a \in \mathbf{a}T \mid ta \in \mathbf{t}T\}$$

For example,

$$S(\varepsilon, SEM_1(a,b)) = \{a\}$$

and

$$S(a, SEM_1(a,b) \mathbin{\mathbf{w}} SEM_1(a,c)) = \{b,c\}$$

The successor set of t consists of all events the mechanism can be involed in after trace t has occurred. Projection, however, can cause the mechanism to refuse some, or all, of the events in the successor set. In the theory of CSP-processes [1],[3] such processes are called nondeterministic. (I would rather call them 'ill-behaved'.) For example, let

$$T = \langle \{a,b,x,y\},\ \{\varepsilon, x, xa, y, yb\} \rangle$$

Then

$$T^-\{a,b\} = \langle \{a,b\},\ \{\varepsilon, a, b\} \rangle$$

By projecting on $\{a,b\}$ symbols x and y have disappeared: they represent internal (non-observable) events. Although

$$S(\varepsilon, T^-\{a,b\}) = \{a,b\}$$

mechanism $T^-\{a,b\}$ may refuse to participate in event a (or b) because internal event y (or x) has already occurred. These types of refusals do not occur if we project on independent alphabets. Alphabet $A \subseteq \mathbf{a}T$ is called *independent* when

$$(\forall t : t \in \mathbf{t}T :\ S(t,T) \subseteq A \Rightarrow S(t,T) = S(t^-A, T^-A))$$

Alphabet $\{a,b\}$ in the example above is not independent:

$$S(y,T) = \{b\}$$

but

$$S(y^-\{a,b\}, T^-\{a,b\}) = S(\varepsilon, T^-\{a,b\}) = \{a,b\}$$

Refusals may also be caused by livelock. For example, let

$$T = \langle \{a, x\},\ \{a, x\}^* \rangle$$

Then

$$T^- a = \langle \{a\},\ \{a\}^* \rangle$$

If internal event x occurs every time it can be chosen, event a never occurs: it is refused forever. Alphabet $A \subseteq \mathbf{a}T$ is called *livelockfree* when

$$(\forall t : t \in \mathbf{t}T : (\exists n : n \geq 0 : (\forall u : u \in A^* \wedge tu \in \mathbf{t}T : \ell(u) \leq n)))$$

where $\ell(u)$ denotes the length of trace u. In the example above alphabets $\{x\}$ and $\{a\}$ are not livelockfree.

Both types of refusals are avoided by projecting on transparent alphabets only. Alphabet $A \subseteq \mathbf{a}T$ is called *transparent* when A is independent and $\mathbf{a}T \backslash A$ (the complement of A within $\mathbf{a}T$) is livelockfree. The importance of transparence is demonstrated by the following theorem [4].

Theorem 1.0 *Let T be a deterministic CSP-process and $A \subseteq \mathbf{a}T$. Then CSP-process $T^- A$ is deterministic if and only if A is transparent.*

Consequently, if we project on transparent alphabets only, no refusals can occur and there is no need to resort to CSP-processes.

2. Data Independence

In this section the events are transmissions of values along channels. Each channel a has a non-empty set $V(a)$ of values that can be transmitted along it. The alphabets of our processes consist of pairs $\langle a, n \rangle$, where $n \in V(a)$. We consider processes such as

$$\begin{aligned} T_0 = &\langle \{a, b\} \times \mathbb{Z} \\ &, pref(\{\langle a, n \rangle \langle b, 2 * n \rangle \mid n \in \mathbb{Z}\}^*) \rangle \end{aligned}$$

where $\mathbb{Z}$ stands for the set of integer numbers. For this process $V(a) = V(b) = \mathbb{Z}$. Process T_0 may informally be described as one that doubles integer numbers.

$$\begin{aligned} T_1 = &\langle \{a, b\} \times \{0, 1\} \\ &, pref(\{\langle a, 0 \rangle,\ \langle a, 1 \rangle \langle b, 1 \rangle\}^*) \rangle \end{aligned}$$

In this case $V(a) = V(b) = \{0, 1\}$. This process may be viewed as one that filters out zeroes and passes on ones. A process that separates zeroes and ones is

$$\begin{aligned} T_2 = &\langle \{a, b, c\} \times \{0, 1\} \\ &, pref(\{\langle a, 0 \rangle \langle b, 0 \rangle,\ \langle a, 1 \rangle \langle c, 1 \rangle\}^*) \rangle \end{aligned}$$

A similar process is

$$T_3 = \langle (\{a\} \times \{0,1\}) \cup (\{b\} \times \{\text{true}, \text{false}\}) \\ , \mathit{pref}(\{\langle a,0\rangle\langle b,\text{true}\rangle, \langle a,1\rangle\langle b,\text{false}\rangle\}^*)\rangle$$

A process that may be viewed as one that arbitrarily permutes two values is

$$T_4 = \langle \{a,b,c\} \times \mathbb{Z} \\ , \mathit{pref}((\{\langle a,m\rangle\langle a,n\rangle\langle b,m\rangle\langle c,n\rangle \mid m \in \mathbb{Z} \wedge n \in \mathbb{Z}\} \\ \cup \{\langle a,m\rangle\langle a,n\rangle\langle b,n\rangle\langle c,m\rangle \mid m \in \mathbb{Z} \wedge n \in \mathbb{Z}\})^*)\rangle$$

If we are interested only in the channels along which the transmissions take place but not in the values transmitted, we replace each symbol $\langle a,n\rangle$ by its channel: $\gamma(\langle a,n\rangle) = a$. Function γ may, of course, be generalized from symbols to sets of symbols, (sets of) traces, and processes. For example

$$\gamma(T_0) = SEM_1(a,b)$$

and

$$\gamma(T_1) = \langle \{a,b\},\ \{a,ab\}^* \rangle$$

Process T is called *data independent* when

$$(\forall t : t \in \mathbf{t}T :\ \gamma(S(t,T)) = S(\gamma(t), \gamma(T)))$$

Set $\gamma(S(t,T))$ consists of all channels along which transmission can take place next. The condition above expresses that this set is independent of the values transmitted thus far. Processes T_0, T_3, and T_4 are data independent and the other two are not. For example,

$$\gamma(S(\langle a,0\rangle, T_1)) = \gamma(\{\langle a,0\rangle, \langle a,1\rangle\}) = \{a\}$$

but

$$S(\gamma(\langle a,0\rangle), \gamma(T_1)) = S(a, \gamma(T_1)) = \{a,b\}$$

In data independent processes we can separate the communication behavior and the computation of the values. (This is sometimes called 'separating data and control.') The *communication behavior* of process T is process $\gamma(T)$. Often the communication behavior is a rather simple process, but many properties, such as transparence, may already be concluded from it.

We shall specify communication behaviors by regular expressions. For example, we specify $\gamma(T_4)$ by the expression $(a\ ;\ a\ ;\ b\ ;\ c)^*$. Notice that semicolons denote concatenation. If the regular expression generates language X the process specified is $\langle A, \mathit{pref}(X)\rangle$, where

A is the set of all symbols that occur in the regular expression. This is a rather primitive way of specifying processes, but it suffices for most of the examples in this paper.

Let $A \subseteq \gamma(\mathbf{a}T)$ and $t \in \mathbf{t}T$. By $t^{-}A$ we mean $t^{-}(\cup a : a \in A : \{a\} \times V(a))$. This definition may be generalized from traces to processes. Then

$$\gamma(T^{-}A) = \gamma(T)^{-}A$$

Data independence is closed under projection on transparent alphabets:

Theorem 2.0 *If T is data independent, $\gamma(\mathbf{a}T)$ is finite, and $A \subseteq \gamma(\mathbf{a}T)$ then*

$$A \text{ transparent} \Rightarrow T^{-}A \text{ data independent}$$

In order to maintain data independence under weaving we have to see to it that whenever a communication along some channel a can take place:

$$a \in \gamma(S(t^{-}\mathbf{a}T, T)) \cap \gamma(S(t^{-}\mathbf{a}U, U))$$

there is actually a transmissible value n:

$$\begin{aligned} &\{\langle a, n\rangle \mid (t^{-}\mathbf{a}T)\langle a, n\rangle \in \mathbf{t}T\} \\ &\cap\{\langle a, n\rangle \mid (t^{-}\mathbf{a}U)\langle a, n\rangle \in \mathbf{t}U\} \neq \phi \end{aligned} \tag{2.0}$$

This is expressed by the following theorem.

Theorem 2.1 *Let T and U be data independent.*
Then

$$\begin{aligned} (\forall t : t \in \mathbf{t}(T \mathbf{w} U) : &\gamma(S(t^{-}\mathbf{a}T, T) \cap S(t^{-}\mathbf{a}U, U)) \\ &= \gamma(S(t^{-}\mathbf{a}T, T)) \cap \gamma(S(t^{-}\mathbf{a}U, U))) \end{aligned} \tag{2.1}$$

if and only if

$$T \mathbf{w} U \text{ data independent and } \gamma(T \mathbf{w} U) = \gamma(T) \mathbf{w} \gamma(U)$$

In order to allow a simple check that (2.1) holds, we partition the channels of each process into *inputs*, *outputs*, and *signals*. We require
(i) for each signal a set $V(a) = \{0\}$;
(ii) each input a (of process T) satisfies

$$\begin{aligned} (\forall t, n : \; &t \in \mathbf{t}T \wedge a \in \gamma(S(t, T)) \wedge n \in V(a) \\ &: \langle a, n\rangle \in S(t, T)) \end{aligned}$$

i.e., T does not restrict the values transmitted along its input channels. In processes T_0 through T_4 we can choose a to be an input and all other channels outputs. (We had this choice in mind in the informal descriptions of these processes.) When weaving we see to the observance of condition (2.1) by requiring that each symbol is an output of at most one of the processes in the weave.

We conclude this section with a—somewhat informal—discussion of a simple example. Its inclusion is meant to show how the computation of output values may be specified.

The example is a process to compute a cumulative sum. Its communication behavior is specified by $(a\,;b)^*$. Channel a is an input, channel b is an output, and $V(a) = V(b) = \mathbb{Z}$. For $t \in \mathrm{t}T$ and $0 \leq i < \ell(t^{\frown}a)$ we let $a(i,t)$ denote the value of the ith transmission along channel a in trace t:

$$\begin{aligned} & a(i,t) = n \\ \equiv & \\ & (\exists u : t = u\langle a, n\rangle : \ \ell(u^{\frown}a) = i) \end{aligned}$$

The values to be computed may then be specified by

$$b(i,t) = (\Sigma j : 0 \leq j \leq i : a(i,t))$$

or, dropping the reference to trace t,

$$b(i) = (\Sigma j : 0 \leq j \leq i : a(i))$$

for $i \geq 0$. Consequently,

$$b(0) = a(0) \tag{2.2}$$

and, for $i \geq 0$,

$$b(i+1) = b(i) + a(i+1) \tag{2.3}$$

We now describe how the output values are computed. To obtain a CSP-like [2] notation we add variables x and y (and assignments) to the communication behavior $(a\,;b)^*$. Our description of the computation is

$$\boxed{y := 0\,;(a?\,x\,;b!\,(y+x)\,;y := y+x)^*} \tag{2.4}$$

The symbols of the communication behavior have been changed into communication statements: as in CSP, each input is postfixed by a question mark and a variable, and each output is postfixed by an exclamation point and an expression. The effect of $b!\,(y+x)$ is that $\langle b, y+x\rangle$ is added to the trace thus far generated, establishing

$$b(\ell(t^{\frown}b)) = y + x$$

Statement $a?\,x$, similarly, establishes

$$a(\ell(t^{\frown}a)) = x$$

Step 0 of the repetition in (2.4) establishes $a(0) = x$, $b(0) = a(0)$ —as required by (2.2)—and $y = b(0)$. Consider, for $i \geq 0$, step $i+1$ of the repetition. We have initially $y = b(i)$. Statement $a?x$ establishes $a(i+1) = x$, statement $b!(y+x)$ establishes $b(i+1) = b(i) + a(i+1)$—as required by (2.3)—and $y := y + x$ establishes $y = b(i+1)$.

3. Conservative Processes

Communication behaviors are often rather simple processes. Checking whether alphabets are transparent is then not difficult. This is in particular the case if the communication behavior is a conservative process.

The successor set $S(t,T)$ consists of all symbols that may follow t. We now introduce the *after set* of t in T, which consists of all *traces* that may follow t:

$$after(t,T) = \{u \in \mathbf{a}T^* \mid tu \in \mathbf{t}T\}$$

for $t \in \mathbf{t}T$. Process T is called *conservative* when

$$\begin{aligned}(\forall t,a,b:\ & a \neq b \wedge ta \in \mathbf{t}T \wedge tb \in \mathbf{t}T \\ & : tab \in \mathbf{t}T \wedge tba \in \mathbf{t}T \\ & \wedge\ after(tab,T) = after(tba,T))\end{aligned}$$

Conservatism expresses, informally speaking, that different events do not disable each other and that the order in which enabled events occur is immaterial.

We have

Theorem 3.0 *For conservative processes T each $A \subseteq \mathbf{a}T$ is independent.*

Conservatism is closed under projection and weaving, as the next two theorems express.

Theorem 3.1 *T conservative $\Rightarrow$ $T^{-}A$ conservative*

Theorem 3.2 *T and U conservative $\Rightarrow$ T **w** U conservative*

The following theorem can be of help to demonstrate conservatism for some simple processes.

Theorem 3.3. *Let R and S be regular expressions consisting of symbols separated by semicolons. Then the process specified by $R\,;\,S^*$ is conservative. Moreover, each subset of its alphabet that contains a symbol occurring in S is transparent.*

For example, the process specified by

$$c\ ;\ d\ ;\ (a\ ;\ a\ ;\ b\ ;\ c)^*$$

is conservative and every non-empty subset of $\{a,b,c\}$ is transparent.

A process may contain *subprocesses*. For reasons of simplicity we restrict ourselves in this paper to processes that have at most one subprocess. We always call the subprocess p. A subprocess has a *type*, which is again a process. If the subprocess has type U we let

$p.U$ denote the process obtained from U by changing all symbols a into $p.a$, read 'p its a'. For example, if

$$U = \langle \{a, b\},\ \{\varepsilon, a, ab\} \rangle$$

then

$$p.U = \langle \{p.a,\ p.b\},\ \{\varepsilon, p.a,\ p.a\ p.b\} \rangle$$

Let process T with $\mathbf{a}T = A$ be specified by a regular expression and let its subprocess have type U. With S denoting the process specified by the regular expression, we require $\mathbf{a}S = A \cup \mathbf{a}p.U$. Then, by definition,

$$T = (S \mathbin{\mathbf{w}} p.U)^{-}A \tag{3.0}$$

For example, let p be of type $SEM_1(a, b)$ and let the regular expression be

$$b\ ;\ (a\ ;\ p.a\ ;\ b\ ;\ p.b)^{*} \tag{3.1}$$

Then $T = SEM_1(b, a)$. However, if p were of type $SEM_1(b, a)$ we would obtain

$$T = \langle \{a, b\},\ \{\varepsilon, b, ba\} \rangle$$

The internal symbols in S represent the channels along which communications with p occur. Each such symbol $p.a$ is an (internal) input or output of S if the corresponding symbol a is an output or input, respectively, of p. This guarantees condition (2.1) for data independence of the weave in (3.0) to hold. Since (3.0) also contains a projection, we have to convince ourselves that A is transparent with respect to S $\mathbf{w}$ $p.U$. For (3.1) this is guaranteed by Theorems 3.2 and 3.3.

An interesting case occurs when T is recursive, i.e., when it has a subprocess of type T. Then (3.0) becomes an equation in T:

$$T = (S \mathbin{\mathbf{w}} p.T)^{-}A$$

By definition process T is the least solution of this equation, where 'least' is meant with respect to the subset order for sets of traces. Phrased differently, process T is the least fixpoint of function f defined by

$$f(x) = (S \mathbin{\mathbf{w}} p.x)^{-}A \tag{3.2}$$

which equals the following least upper bound [4]:

$$(\mathbf{LUB}\, i : i \geq 0 : f^{i}(\langle A, \{\varepsilon\} \rangle))$$

For example, if S is

$$(a\,;\,p.a\,;\,b\,;\,p.b)^* \tag{3.3}$$

or

$$a\,;\,b\,;\,(p.a\,;\,a\,;\,b\,;\,p.b)^* \tag{3.4}$$

we have $T = SEM_1(a,b)$. However, if S is

$$(a\,;\,p.a\,;\,p.b\,;\,b)^*$$

we find

$$T = \langle \{a,b\},\ \{\varepsilon, a\}\rangle$$

Theorem 3.4 *For conservative S the least fixpoint of f, as defined in (3.2), is conservative.*

For some processes, for example, those specified by regular expressions conforming to Theorem 3.3, it is sensible to talk about the duration between (external) events, or, more precisely, about the number of ordered internal events between successive external events. Let

$$T = (S\ \mathbf{w}\ p.U)^{-}A$$

A sequence function

$$\sigma : \mathbf{a}S \times \mathbb{N} \to \mathbb{N}$$

($\mathbb{N}$ the set of natural numbers) is a function satisfying

$$\begin{aligned}(\forall t, a, b, :\ & tab \in \mathbf{t}S \wedge a \in \mathbf{a}S \wedge b \in \mathbf{a}S \\ & :\ \sigma(a, \ell(t^{-}a)) < \sigma(b, \ell(ta^{-}b)))\end{aligned}$$

We require that subprocess p has a corresponding sequence function σ', i.e., one that satisfies

$$(\forall a, i :\ a \in \mathbf{a}U \wedge i \geq 0 :\ \sigma(p.a, i) = \sigma'(a, i))$$

If σ is a sequence function then so is $\sigma + m$ for all natural m.

The process specified by (3.3) has, for example, the following sequence function:

$$\begin{aligned} \sigma(a,i) &= 4 * i \\ \sigma(b,i) &= 4 * i + 2 \\ \sigma(p.a,i) &= 4 * i + 1 \\ \sigma(p.b,i) &= 4 * i + 3 \end{aligned} \tag{3.5}$$

This is an allowed sequence function, since $\sigma(p.a,i) = \sigma(a,i) + 1$, $\sigma(p.b,i) = \sigma(b,i) + 1$, and $\sigma + 1$ is a sequence function for p.

We say that process T has *constant response time* when there exists a sequence function σ for T such that

$$\begin{aligned} (\exists n : n \geq 1 : (\forall t, a, b : tab \in \mathrm{t}T \wedge a \in \mathrm{a}T \wedge b \in \mathrm{a}T \\ : \sigma(b, \ell(ta^{-}b)) - \sigma(a, \ell(t^{-}a)) \leq n)) \end{aligned}$$

The process specified by (3.3) has constant response time: for σ as given in (3.5) the condition above holds for $n = 2$. The process specified by (3.4) does *not* have constant response time. A possible sequence function for that process is

$$\begin{aligned} \sigma(a,i) &= (i+1)^2 - 1 \\ \sigma(b,i) &= (i+1)^2 \\ \sigma(p.a,i) &= (i+2)^2 - 2 \\ \sigma(p.b,i) &= (i+2)^2 + 1 \end{aligned}$$

We have now assembled all the theory we need to discuss a number of interesting examples. These are presented in the next three sections.

4. Polynomial Multiplication

Given is a polynomial q of degree M, $M \geq 0$. For $0 \leq i \leq M$ we let q_i denote the coefficient of x^i in q:

$$q = q_M * x^M + \cdots + q_1 * x + q_0$$

Polynomial q has to be multiplied by a polynomial r of degree N, $N \geq 0$, yielding a polynomial s of degree $M + N$, given by

$$s_{M+N-i} = (\Sigma j : \max(i - M, 0) \leq j \leq \min(i, N) : q_{M+j-i} * r_{N-j})$$

for $0 \leq i \leq M + N$.

The process that carries out the multiplication is to have input a and output b with $V(a) = V(b) = \mathbb{Z}$. Along input a the coefficients r_i are transmitted in order of decreasing indices, followed by zeroes:

$$a(i) = \begin{cases} r_{N-i} & \text{if } 0 \leq i \leq N \\ 0 & \text{if } i > N \end{cases} \tag{4.0}$$

Along output b the coefficients of s are to be transmitted, followed by zeroes:

$$b(i) = \begin{cases} s_{M+N-i} & \text{if } 0 \leq i \leq M+N \\ 0 & \text{if } i > M+N \end{cases}$$

The communication behavior is $(a\,;\,b)^*$. In view of (4.0) we have for $0 \leq i \leq M+N$

$$b(i) = (\Sigma j : \max(i-M, 0) \leq j \leq \min(i, N) : q_{M+j-i} * a(j)) \tag{4.1}$$

We design for $0 \leq k \leq M$ processes MUL_k, that have (external) communication behavior $(a\ ;\ b)^*$ and, cf. (4.1),

$$b(i) = (\Sigma j : \max(i-k, 0) \leq j \leq \min(i, N) : q_{k+j-i} * a(j)) \tag{4.2}$$

if $0 \leq i \leq k+N$, and $b(i) = 0$ if $i > k+N$. Then MUL_M is the process we are interested in.

Process MUL_0 is simple: (4.2) yields for $k = 0$

$$b(i) = \begin{cases} q_0 * a(i) & \text{if } 0 \leq i \leq N \\ 0 & \text{if } i > N \end{cases}$$

Since $a(i) = 0$ for $i > N$, this may be simplified to

$$b(i) = q_0 * a(i)$$

for $i \geq 0$. The computation of MUL_0 may be specified by

$$\boxed{(a?\,x\ ;\ b!\,(q_0 * x))^*}$$

We now turn to MUL_k for $1 \leq k \leq M$. It has a subprocess of type MUL_{k-1}. Consequently,

$$p.a(i) = a(i) \qquad \text{for } i \geq 0 \tag{4.3}$$

$$p.b(i) = \begin{cases} (\Sigma j : \max(i-k+1, 0) \leq j \leq \min(i, N) & \\ \qquad : q_{k-1+j-i} * a(j)) & \text{if } 0 \leq i < k+N \\ 0 & \text{if } i \geq k+N \end{cases} \tag{4.4, 4.5}$$

By (4.2)

$$b(0) = q_k * a(0) \tag{4.6}$$

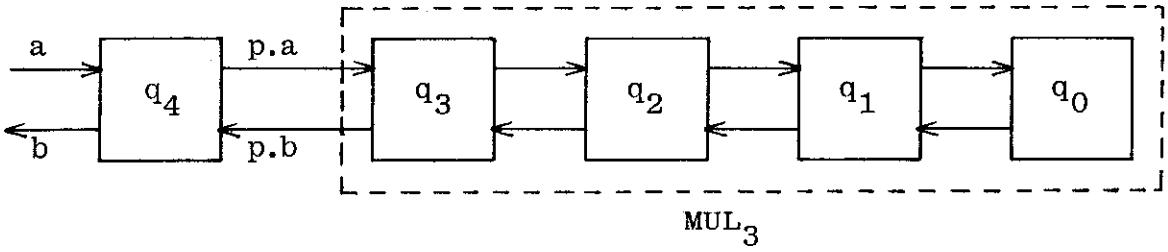

Fig. 1. Process MUL_4

and for $0 \leq i < k + N$

$$b(i+1) = (\Sigma j : \max(i-k+1, 0) \leq j \leq \min(i+1, N) : q_{k-1+j-i} * a(j))$$

Hence, by (4.4),

$$b(i+1) = \begin{cases} p.b(i) & \text{if } N \leq i < k+N \\ p.b(i) + q_k * a(i+1) & \text{if } 0 \leq i < N \end{cases}$$

Since $a(i+1) = 0$ for $i \geq N$, this may be simplified to

$$b(i+1) = p.b(i) + q_k * a(i+1)$$

for $0 \leq i < k + N$. For $i \geq k + N$ we have $p.b(i) = 0$ and $a(i+1) = 0$. Consequently,

$$b(i+1) = p.b(i) + q_k * a(i+1) \tag{4.7}$$

for $i \geq 0$.

Relations (4.3), (4.6), and (4.7) tell us how the output values may be computed. We choose

$$(a\,;\, p.a\,;\, b\,;\, p.b)^* \tag{4.8}$$

as the communication behavior. Then $p.a(i)$ follows $a(i)$, as required by (4.3), $b(0)$ follows $a(0)$, as required by (4.6), and $b(i+1)$ follows $p.b(i)$ and $a(i+1)$, as required by (4.7). According to Theorem 3.3 alphabet $\{a, b\}$ is transparent. We have already shown that the process has constant response time and that, with S denoting the process specified by (4.8),

$$(S \text{ w } SEM_1(a,b))^{-}\{a,b\} = SEM_1(a,b)$$

Given (4.3), (4.6), and (4.7), it is now simple to specify the computation of the output values:

$$\boxed{y := 0\ ;\ (a?\,x\ ;\ p.a!\,x\ ;\ b!\,(y + q_k * x)\ ;\ p.b?\,y)^*} \tag{4.9}$$

Process MUL_k consists of the process specified by (4.9), which uses value q_k, and MUL_{k-1} as a subprocess. Figure 1 shows process MUL_4, in which each process that uses value q_k is drawn as a rectangle with q_k in it.

We have designed an array of $M+1$ cells. Each cell stores one coefficient of polynomial q. All cells are equal, except for the last one, which has no right neighbor. (We could have made this one equal by adding a cell at the end that returns value 0 upon every input.)

The coefficients of polynomials r and s are transmitted in order of decreasing indices. This order is actually immaterial. We could have done the same analysis for the reverse order, and the only change would have been to replace q_k in process MUL_k by q_{M-k}: the order in which the coefficients of q are distributed over the cells is reversed as well.

Our solution is independent of the degree of r. Process MUL_M will multiply polynomial q by polynomials of any degree. In order for the complete product to be produced at b, we have to require of the input only that the coefficients of r are followed by at least M zeroes. But afterwards we have $y = 0$ and a new polynomial may be input again! We have thus designed a systolic computation for repeatedly multiplying a fixed polynomial q by other polynomials. The only restriction on the input is that the coefficients of different polynomials are separated by (at least) M zeroes.

5. Cyclic Encoding

A nice application of polynomial multiplication is the encoding of messages, using a cyclic code. Given is a polynomial q of degree M with $M \geq 1$, $q_i \in \{0,1\}$, and $q_M = 1$. Polynomial q is often called the generator polynomial of the cyclic code. Each message is a sequence $r_N r_{N-1} \ldots r_0$, where $r_i \in \{0,1\}$ and $N \geq 0$. The message may be regarded as the coefficients of a polynomial r. The encoded message consists of the coefficients of polynomial

$$r * x^M \oplus t \tag{5.0}$$

of degree $M+N$, where t is the remainder polynomial after division of $r * x^M$ by q and $\oplus$ denotes addition modulo 2. More precisely, polynomial t is defined by

$$r * x^M = q * d \oplus t \tag{5.1}$$

where d is a polynomial of degree N and t has degree $M-1$.

For example, if the generator polynomial is $x^4 + x + 1$ ($M = 4$) and the message is 101110111, i.e., $r = x^8 + x^6 + x^5 + x^4 + x^2 + x + 1$, we find for (5.1)

$$\begin{aligned} & x^{12} + x^{10} + x^9 + x^8 + x^6 + x^5 + x^4 \\ = \\ & (x^4 + x + 1) * (x^8 + x^6 + x^3 + x) \oplus (x^3 + x^2 + x) \end{aligned}$$

The encoded message is, by (5.0), 1011101111110: the sequence 1110 of M check bits, corresponding to polynomial $x^3 + x^2 + x$, has been added to the message. A well-known example is the use of $x + 1$ as generator polynomial. It results in adding a parity bit.

On account of (5.1), polynomial (5.0), representing the encoded message, equals $q * d$. Our encoder, consequently, has to multiply polynomials q and d, a problem we have already solved in Section 4. Polynomial d is, of course, not given, but the amazing property—pointed out to me by F. W. Sijstermans of Philips Research—is that the coefficients of d may be determined as polynomial r is input.

By (5.1) we conclude

$$d_N = r_N \tag{5.2}$$

Notice that for $0 \leq j \leq N$

$$(q * d)_{M+j} = (r * x^M \oplus t)_{M+j} = (r * x^M)_{M+j} = r_j \tag{5.3}$$

Let q' be a polynomial of degree $M-1$ such that

$$q = x^M \oplus q' \tag{5.4}$$

Then, for $0 \leq j < N$,

$$\begin{aligned} d_j &= (x^M * d)_{M+j} \\ &= \{\text{by (5.4)}\} \quad ((q \oplus q') * d)_{M+j} \\ &= (q * d)_{M+j} \oplus (q' * d)_{M+j} \\ &= \{\text{by (5.3)}\} \quad r_j \oplus (q' * d)_{M+j} \end{aligned} \tag{5.5}$$

We introduce a subcomponent p of type MUL_{M-1} that computes $q' * d$. The output of p can be used to determine both d_j for $0 \leq j < N$ and, as will turn out later, $b(i)$ for $N < i \leq N + M$.

Our process has input a and output b. For $0 \leq i \leq N$

$$a(i) = r_{N-i} \tag{5.6}$$

and for $0 \leq i \leq M + N$

$$b(i) = (q * d)_{M+N-i}$$

The external communication behavior is

$$(a\,;\, b)^{N+1}\,;\, b^M$$

where S^N denotes N concatenations of S, for example

$$(a\,;\, b)^2 = (a\,;\, b\,;\, a\,;\, b)$$

We suggest to insert the internal communications as follows:

$$(a\,;\,p.a\,;\,b\,;\,p.b)^{N+1}\,;\,(p.a\,;\,b\,;\,p.b)^{M}$$

We have, according to the specification of MUL_{M-1},

$$p.a(i) = \begin{cases} d_{N-i} & \text{if } 0 \le i \le N \\ 0 & \text{if } N < i \le M+N \end{cases} \tag{5.7}$$

and

$$p.b(i) = \begin{cases} (q' * d)_{M+N-i-1} & \text{if } 0 \le i < M+N \\ 0 & \text{if } i = M+N \end{cases} \tag{5.8}$$

For $0 \le i \le N$ we have, by (5.3),

$$b(i) = (q * d)_{M+N-i} = r_{N-i} = a(i)$$

For $N \le i < M+N$ we find

$$\begin{aligned} b(i+1) &= (q * d)_{M+N-i-1} \\ &= \{\text{by (5.4)}\} \quad ((x^M \oplus q') * d)_{M+N-i-1} \\ &= (x^M * d)_{M+N-i-1} \oplus (q' * d)_{M+N-i-1} \\ &= \{M+N-i-1 \le M-1\}\ (q' * d)_{M+N-i-1} \\ &= \{\text{by (5.8)}\}\ p.b(i) \end{aligned}$$

Outputs $p.a(i)$ may be determined as follows.

$$\begin{aligned} p.a(0) &= \{\text{by (5.7)}\}\ d_N \\ &= \{\text{by (5.2)}\}\ r_N \\ &= \{\text{by (5.6)}\}\ a(0) \end{aligned}$$

For $0 \le i < N$ we derive

$$\begin{aligned} p.a(i+1) &= \{\text{by (5.7)}\}\ d_{N-i-1} \\ &= \{\text{by (5.5)}\}\ r_{N-i-1} \oplus (q' * d)_{M+N-i-1} \\ &= \{\text{by (5.6) and (5.8)}\}\ a(i+1) \oplus p.b(i) \end{aligned}$$

Furthermore, $p.a(i) = 0$ for $N < i \le M+N$.

Summarizing, we have

$$b(i) = \begin{cases} a(i) & \text{if } 0 \le i \le N \\ p.b(i-1) & \text{if } N < i \le M+N \end{cases}$$

and

$$p.a(i) = \begin{cases} a(i) & \text{if } i = 0 \\ a(i) \oplus p.b(i-1) & \text{if } 1 \leq i \leq N \\ 0 & \text{if } N < i \leq M + N \end{cases}$$

The computation of these output values may be specified as follows.

$$\begin{array}{l} y := 0 \\ ;(a?\,x\ ;\ p.a!\,(x \oplus y)\ ;\ b!\,x\ ;\ p.b?\,y)^{N+1} \\ ;(p.a!\,0\ ;\ b!\,y\ ;\ p.b?\,y)^{M} \end{array}$$

Since $p.b(M+N) = 0$, the last statement reestablishes $y = 0$. We have, therefore, a simple way of changing the program into one that repeatedly encodes messages:

$$\begin{array}{l} y := 0 \\ ;((a?\,x\ ;\ p.a!\,(x \oplus y)\ ;\ b!\,x\ ;\ p.b?\,y)^{N+1} \\ \quad ;\ (p.a!\,0\ ;\ b!\,y\ ;\ p.b?\,y)^{M} \\ )* \end{array}$$

Making the process independent of the message length requires a message-separator signal. Calling that signal c, our solution becomes

$$\begin{array}{l} y := 0 \\ ;((a?\,x\ ;\ p.a!\,(x \oplus y)\ ;\ b!\,x\ ;\ p.b?\,y)^{*} \\ \quad ;\ c\ ;\ (p.a!\,0\ ;\ b!\,y\ ;\ p.b?\,y)^{M} \\ )* \end{array}$$

By adding one cell to MUL_{M-1} (or one could say, by changing the first cell of MUL_M) we have obtained a systolic computation for encoding messages of varying lengths. Figure 2 shows a drawing of the process.

The communication behavior at the source side is $(a^*\,;\,c)^*$: the messages are separated by signal c, and the value of M is immaterial to the source side transmissions.

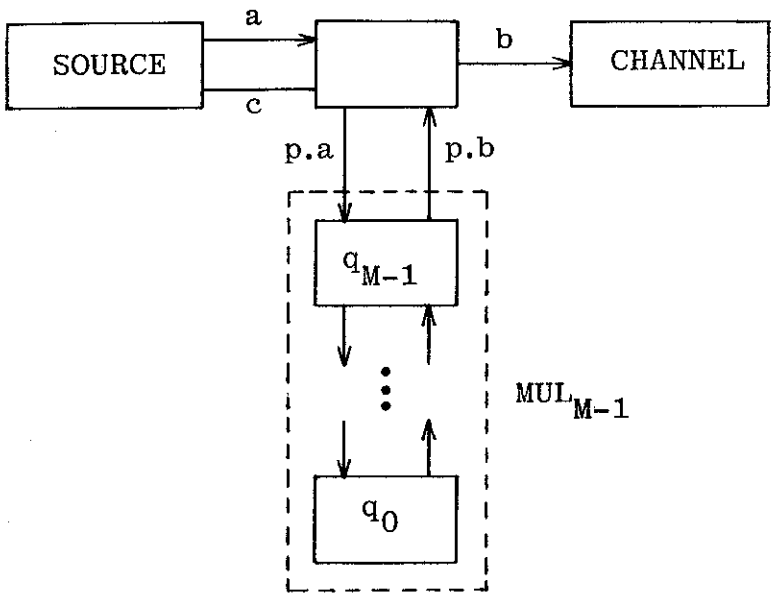

Fig. 2. Cyclic encoder

6. Palindrome Recognition

In this section we discuss a recursive palindrome recogngizer. The object is to specify a process with external behavior $(b; a)^*$, where b is output and a is input, $V(b) = \{\text{true}, \text{false}\}$, and $V(a) = \mathbb{Z}$. The value of output b has to indicate whether the sequence thus far received at input a is a palindrome. More precisely, for $i \geq 0$

$$b(i) = (\forall i : 0 \leq j < i : a(j) = a(i-1-j))$$

We have

$$b(0) = b(1) = \text{true} \tag{6.0}$$

and for $i \geq 0$

$$\begin{aligned} b(i+2) &= (\forall j : 0 \leq j < i+2 : a(j) = a(i+1-j)) \\ &= (a(0) = a(i+1)) \\ &\quad \wedge (\forall j : 1 \leq j < i+1 : a(j) = a(i+1-j)) \\ &\quad \wedge (a(i+1) = a(0)) \\ &= (a(0) = a(i+1)) \\ &\quad \wedge (\forall j : 0 \leq j < i : a(j+1) = a(i-j)) \end{aligned} \tag{6.1}$$

The latter conjunct is again the outcome of a palindrome recognizer, but now one that pertains to the input sequence beginning at $a(1)$. We, therefore, introduce a subprocess of the same type as the process we are designing: for $i \geq 0$

$$p.a(i) = a(i+1) \tag{6.2}$$

and

$$p.b(i) = (\forall j : 0 \leq j < i : p.a(j) = p.a(i-1-j))$$

Using (6.2), the latter relation may be written as

$$p.b(i) = (\forall j : 0 \leq j < i : a(j+1) = a(i-j))$$

By (6.1) we then find for $i \geq 0$

$$b(i+2) = (a(0) = a(i+1)) \wedge p.b(i) \tag{6.3}$$

Since the first two outputs at b are computed differently from the subsequent ones, we suggest

$$b\,;\, a\,;\, b\,;\, (a\,;\, p.b\,;\, b\,;\, p.a)^* \tag{6.4}$$

as the communication behavior. Then $p.a(i)$ follows $a(i+1)$, cf. (6.2), and $b(i+2)$ follows $a(i+1)$ and $p.b(i)$, as required by (6.3). By Theorem 3.3 alphabet $\{a, b\}$ is transparent. With S denoting the process specified by (6.4) we have

$$(S \mathbin{\mathbf{w}} p.S)^{-}\{a, b\} = SEM_1(b, a)$$

This shows that the process has the required external behavior. It also has constant response time. A sequence function is

$$\sigma(a, i) = 4 * i + 2$$
$$\sigma(b, i) = 4 * i$$
$$\sigma(p.a, i) = 4 * i + 9$$
$$\sigma(p.b, i) = 4 * i + 7$$

This is an allowed sequence function, since $\sigma + 7$ is a sequence function for p and

$$\sigma(p.a, i) = \sigma(a, i) + 7$$
$$\sigma(p.b, i) = \sigma(b, i) + 7$$

Given (6.0), (6.2), and (6.3), it is not difficult to extend (6.4) with the computation of the output values:

$$\begin{array}{|l|}\hline b!\,\mathrm{true}\ ;\ a?\,z\ ;\ b!\,\mathrm{true} \\ ;(a?\,x\ ;\ p.b?\,y\ ;\ b!\,((z = x) \wedge y)\ ;\ p.a!\,x \\ \hline\end{array}$$

(Notice that $a?\,z$ establishes $a(0) = z$.)

We have designed an infinite array of cells. With the sequence function given above, subprocess p starts at moment $\sigma(p.b, 0)$, i.e., at moment 7. In general, cell k, for $k \geq 0$, starts at moment $7 * k$. Cell 0 produces the answers. For $i \geq 0$ answer $b(i)$ is produced at moment $\sigma(b, i) = 4 * i$. At that moment all cells k for which $7 * k \leq 4 * i$ have started: slightly over $i/2$ cells are required to produce answer $b(i)$.

7. Conclusion

Systolic arrays are often presented and explained by means of pictures. We have refrained from doing so. Of course, we showed a few pictures, but they were merely used as illustrations: in no way did our discussion rely on them.

We discussed systolic computations in terms of their input/output behaviors. This is a method for which the formalism of trace theory is very well-suited. We have isolated the concepts of data independence, transparence, and conservatism as central notions in the study of systolic computations. We are pleased with the nice way in which these concepts tie together. In contrast to what is customary, we did not describe the computations in terms of global states. As a matter of fact, we suspect that these solutions would not have been found then: in [9] the palindrome recognizer requires cells that are slightly more complicated (essentially, the combination of two of ours) to achieve that communication takes place with the neighbor cells only.

One of the reasons why we want each cell to have a fixed number of neighbor cells is to facilitate the realization of our computations as VLSI circuits. The main reason to have all synchronization be accomplished by message passing is that we would like these VLSI circuits to be delay-insensitive [10], which excludes the use of global clocks. The work by Alain J. Martin on compiling CSP-like programs into delay-insensitive VLSI circuits [6] shows that such realizations may be obtained by introducing handshaking protocols to implement the communication actions.

8. Acknowledgements

Acknowledgements are due to F. W. Sijstermans of Philips Research who pointed out to me the relation between cyclic encoding and polynomial multiplication. My solution for the latter is a slight variation of his.

I am very grateful to Gerard Zwaan, Anne Kaldewaij, and Tom Verhoeff for sharing with me their ideas on data independence, transparence, and conservatism. Many results presented in Sections 1, 2, and 3 are due to their prolific efforts. The discussions with them and with the other members of the Eindhoven VLSI Club have been a great help to me.

I thank Roland C. Backhouse and Jan L. A. van de Snepscheut for directing my attention to the problems of palindrome recognition and cyclic encoding.

California Institute of Technology is acknowledged for giving me the opportunity to prepare this paper during my visit of winter 1986-87.

The research described in this paper was in part sponsored by the Defense Advanced Research Projects Agency, ARPA Order number 3771, and was monitored by the Office of Naval Research under contract number N00014-79-C-0597.

9. References

[1] Brookes, S.D., Roscoe, A.W. An improved failures model for communicating processes. In: Seminar on Concurrency (S.D. Brookes, A.W. Roscoe, G. Winskel, eds.). Springer, Berlin, 1985 (Lecture Notes in Computer Science: 197), 281–305.

[2] Hoare, C.A.R. Communicating sequential processes. Comm. ACM **21** (1978), 666–677.

[3] Hoare, C.A.R. Communicating Sequential Processes. Prentice Hall, New York, 1985.

[4] Kaldewaij, A. A Formalism for Concurrent Processes. Doctoral Dissertation, Eindhoven University of Technology, Eindhoven, 1986.

[5] Kung, H.T. Let's design algorithms for VLSI systems. In: Proc. 1st Caltech Conference (C.L. Seitz, ed.). California Institute of Technology, Pasadena, 1979, 65–90.

[6] Martin, A.J. Compiling communicating processes into delay-insensitive VLSI circuits. Distributed Computing **1** (1986), 226–234.

[7] Rem, M. Concurrent computations and VLSI circuits. In: Control Flow and Data Flow: Concepts in Distributed Programs (M. Broy, ed.). Springer, Berlin, 1985, 399–437.

[8] Snepscheut, J.L.A. van de. Trace Theory and VLSI Design. Springer, Berlin, 1985 (Lecture Notes in Computer Science: 200).

[9] Snepscheut, J.L.A. van de, Swenker, J. On the design of some systolic algorithms. Note JAN 131, Groningen University, Groningen, 1986.

[10] Udding, J.T. A formal model for defining and classifying delay-insensitive circuits and systems. Distributed Computing **1** (1986), 197–204.